**String**

String handling is a way of handling and manipulating strings in java with the help of lot concepts like concatenation, comparison etc.

***String:***

***In General*** – String is sequence of characters.  
***In Java*** – String is an object which is created by using String class. String objects are immutable and they can’t modified i.e. any change in the existing object will result into a new object.

***How to create string object?***

1. **By String literal.**
2. **By new keyword.**

***1. By String literal:***

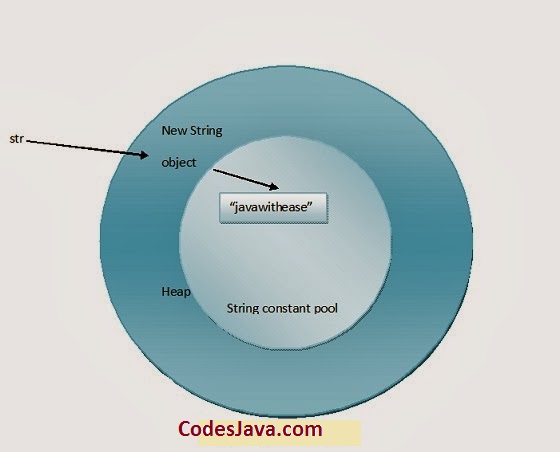
***String literal:*** A sequence of characters enclosed in double quotes. String literal is a concept of java language. It is not an object from java.lang.String.  e.g. – “Hello java”.  In the string literal case one object and one reference variable is created.  Object is placed in string constant pool.  
***String literal/constant pool:*** is a special part of heap memory used to store string literals or string constants.  
Every time a literal is created, JVM checks the string constant pool for it. If string literal is already in the pool then no new object will be created in the pool, a reference of the already existing object will returns. If string literal is not exist in the string constant pool then new instance will be created and placed in the string constant pool.  
E.g. – String str1 = “javawithease”  
String str2 = “javawithease”

  
   
In the above example in case of str1 JVM checks for “javawithease” in string constant pool. First time it will be not in the pool hence JVM create new instance and placed it into the pool. In case of str2 JVM will find the “javawithease” in the pool. Hence no new instance will be created and reference of the already existing instance will be return.

***Java uses the concept of java literal to increase memory efficiency. Because, no new instance will be created for a string literal if it is already exist in string constant pool.***

***2. By new keyword:***

String objects can be created with new keyword also. In this case two objects and one reference variable is created. One object is created in heap area (non-pool) and other (string literal) is placed in string constant pool. The variable will refer to the object in heap area.  
e.g.- String str = new String(“javawithease”).



***Why string objects are immutable in java?***

As we discussed above java uses the concept of String literal. Suppose n reference variables refer to one object “javawithease” .If one reference variable changes the value of the object, it will be affected to all other n-1 reference variables. That’s why string objects are immutable in java.

In java there are three ways to compare two strings.

## Ways of String Comparison:

**1. By == operator.**  
**2. By equals() method.**  
**3. By compareTo() method.**

## 1. By == operator:

== operator compares the references of the string objects not the actual content of the string objects. It returns true if reference of the compared strings are equal, otherwise returns false.

## Example:

**StringComparisonExample1.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of == operator.***  ***\* @author codesjava.***  ***\*/***  **class** TestString{  String str1 = "codesjava";  String str2 = "codesjava";  String str3 = **new** String("codesjava");    ***/\*\****  ***\* This method is used to compare strings using == operator.***  ***\* @author codesjava***  ***\*/***  **public** **void** stringComparison(){  *//return true, because str1 and str2 both refers to the*  *//same instance created in String constant pool.*  System.out.println(str1 == str2);    *//return false, because str3 refers to the*  *// instance created in nonpool.*  System.out.println(str1 == str3);  }  }    **public** **class** StringComparisonExample1 {  **public** **static** **void** main(String args[]){  *//creating TestString object.*  TestString obj = **new** TestString();    *//method call*  obj.stringComparison();  }  } |

## Output:

|  |
| --- |
| **True**  **false** |

[***Download this example.***](https://codesjava.com/wp-content/uploads/2014/08/StringExample1.rar)

## 2. By equals() method:

equals method compares the actual content of string objects not the references. equals() is an object class method and string class overrides it. String class provides following format of equals() method.

**a. public boolean equals(Object obj):** Compares the string with the specified object. Returns true if actual content are equal, otherwise returns false.  
**b. public Boolean equalsIgnoreCase(String str):** Compares the actual content of the string with the content of specified string. Returns true if two string’s content are equal ignoring case, otherwise returns false.

## Example:

**StringComparisonExample2.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of equals method.***  ***\* @author codesjava.***  ***\*/***  **class** TestString{  String str1 = "codesjava";  String str2 = "codesjava";  String str3 = **new** String("codesjava");  String str4 = "jai";  String str5 = "CODESJAVA";    ***/\*\****  ***\* This method is used to compare strings using equals operator.***  ***\* @author codesjava***  ***\*/***  **public** **void** stringComparison(){  *//return true, because content are same.*  System.out.println(str1.equals(str2));  System.out.println(str2.equals(str3));    *//return false, because content are not same.*  System.out.println(str2.equals(str4));    *//return false, because content are not same*  *//(differ in case).*  System.out.println(str2.equals(str5));    *//return true, because content are same ignoring case.*  System.out.println(str2.equalsIgnoreCase(str5));  }  }    **public** **class** StringComparisonExapmle2 {  **public** **static** **void** main(String args[]){  *//creating TestString object.*  TestString obj = **new** TestString();    *//method call*  obj.stringComparison();  }  } |

## Output:

|  |
| --- |
| **true**  **true**  **false**  **false**  **true** |

[***Download this example.***](https://codesjava.com/wp-content/uploads/2014/08/StringExample2.rar)

## 3. By compareTo():

compareTo() method compares the two strings lexicographically i.e. character by character. String class provides the following formats of compareTo() method.

**a. public int compareTo(String str)**  
**b. public int compareToIgnoreCase(String str)**

It returns 0 if the argument string is equal to this string, less than 0 if this string is lexicographically less than the string argument and greater than 0 if this string is lexicographically greater than the string argument.

## Example:

**StringComparisonExample3.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of compareTo method.***  ***\* @author codesjava.***  ***\*/***  **class** TestString{  String str1 = "codesjava";  String str2 = "codesjava";  String str3 = "jai";  String str4 = "sandy";    ***/\*\****  ***\* This method is used to compare***  ***\* strings using compareTo operator.***  ***\* @author codesjava***  ***\*/***  **public** **void** stringComparison(){  *//return 0, because content are same.*  System.out.println(str1.compareTo(str2));    *//return greater than 0, because str1*  *//lexicographically greater than str2.*  System.out.println(str1.compareTo(str3));    *//return less than 0, because str1*  *//lexicographically less than str2.*  System.out.println(str3.compareTo(str4));  }  }    **public** **class** StringComparisonExample3 {  **public** **static** **void** main(String args[]){  *//creating TestString object.*  TestString obj = **new** TestString();    *//method call*  obj.stringComparison();  }  } |

## Output:

|  |
| --- |
| 0  13  -9 |

**intern():**Returns a canonical representation for the string object.

**Syntax:** **public String intern().**

##### *****Note: It returns a string that has the same contents as this string, but is guaranteed to be from a pool of unique strings.*****

## Example:

**StringInternExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of intern() method.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str1 = "www.codesjava.com";  String str2;    ***/\*\****  ***\* This method is used to show the use of intern() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** internTest(){  str2 = str1.intern();  System.out.println(str2);  }  }    **public** **class** StringInternExample {  **public** **static** **void** main(String args[]){  *//creating TestString object*  TestString obj = **new** TestString();    *//method call*  obj.internTest();  }  } |

## Output:

|  |
| --- |
| www.codesjava.com |

# String Interning — What, Why, and When?

### **Learn about string interning, a method of storing only one copy of each distinct string value, which must be immutable.**

**What is String Interning?**

String Interning is a method of storing only one copy of each distinct String Value, which must be immutable.

In Java, String class has a public method intern() that returns a canonical representation for the string object. Java’s String class privately maintains a pool of strings, where String literals are automatically interned.

**When the *intern()* method is invoked on a *String* object it looks the string contained by this *String* object in the pool, if the string is found there then the string from the pool is returned. Otherwise, this *String* object is added to the pool and a reference to this *String* object is returned.**

The intern() method helps in comparing two String objects with == operator by looking into the pre-existing pool of string literals, no doubt it is faster than equals() method. The pool of strings in Java is maintained for saving space and for faster comparisons.Normally Java programmers are advised to use equals(), not ==, to compare two strings. This is because == operator compares memory locations, while equals() method compares the content stored in two objects.

**Why and When to Intern ?**

Though Java automatically interns all Strings by default, remember that we only need to intern strings when they are not constants, and we want to be able to quickly compare them to other interned strings. The intern() method should be used on strings constructed with new String() in order to compare them by == operator.

Let’s take a look at the following Java program to understand the **intern()**behavior.

public class TestString {

public static void main(String[] args) {

String s1 = "Test";

String s2 = "Test";

String s3 = new String("Test");

final String s4 = s3.intern();

System.out.println(s1 == s2);

System.out.println(s2 == s3);

System.out.println(s3 == s4);

System.out.println(s1 == s3);

System.out.println(s1 == s4);

System.out.println(s1.equals(s2));

System.out.println(s2.equals(s3));

System.out.println(s3.equals(s4));

System.out.println(s1.equals(s4));

System.out.println(s1.equals(s3));

}

}

//Output

true

false

false

false

true

true

true

true

true

true

**toLowerCase():** Converts all of the characters in this String to lower case.

**Syntax:** **public String toLowerCase().**

**toUpperCase():** Converts all of the characters in this String to upper case.

**Synatx: public String toUpperCase().**

## Example:

**CaseChangeExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of***  ***\* toUpperCase and toLowerCase methods.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = "Jai";    ***/\*\****  ***\* This method is used to show the use of toUpperCase() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** upperCase(){  *//will convert all characters in upper case.*  System.out.println(str.toUpperCase());  }    ***/\*\****  ***\* This method is used to show the use of toLowerCase() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** lowerCase(){  *//will convert all characters in lower case.*  System.out.println(str.toLowerCase());  }  }    **public** **class** CaseChangeExample {  **public** **static** **void** main(String args[]){  *//creating TestString object*  TestString obj = **new** TestString();    *//method call*  obj.upperCase();  obj.lowerCase();  }  } |

## Output:

|  |
| --- |
| JAI  jai |

**length():**Returns the length of this string.

**Syntax:** **public int length().**

## Example:

**StringLengthExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of length() method.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = "www.codesjava.com";    ***/\*\****  ***\* This method is used to show the use of length() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** stringLengthTest(){  System.out.println(str.length());  }  }    **public** **class** StringLengthExample {  **public** **static** **void** main(String args[]){  *//creating TestString object*  TestString obj = **new** TestString();    *//method call*  obj.stringLengthTest();  }  } |

## Output:

|  |
| --- |
| 17 |

**trim():**Returns a copy of the string, with leading and trailing whitespace omitted.

**Syntax:** **public String trim().**

## Example:

**StringTrimExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of trim() method.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = " www.codesjava.com ";    ***/\*\****  ***\* This method is used to show the use of trim() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** trimString(){  *//will remove all leading and trailing whitespace.*  System.out.println(str.trim());  }  }    **public** **class** StringTrimExample {  **public** **static** **void** main(String args[]){  *//creating TestString object*  TestString obj = **new** TestString();    *//method call*  obj.trimString();  }  } |

## Output:

|  |
| --- |
| www.codesjava.com |

**indexOf(String str):** Returns the index of the first occurrence of the specified substring within this string.

**Syntax:**  **public int indexOf(String str).**

##### Note: If no such occurrence of substring within this string than it returns -1.

**lastIndexOf(String str):** Returns the index of the last occurrence of the specified substring within this string.

**Syntax:** **public int lastIndexOf(String str).**

##### Note: If no such occurrence of substring within this string than it returns -1.

## Example:

**StringIndexTestExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use***  ***\* of indexOf and lastIndexOf() method.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = "www.codesjava.com";    ***/\*\****  ***\* This method is used to show the use of indexOf() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** indexOfTest(){  System.out.println(str.indexOf("o"));  }    ***/\*\****  ***\* This method is used to show the use of lastIndexOf() method.***  ***\* @author codesjava***  ***\*/***  **public** **void** lastIndexOfTest(){  System.out.println(str.lastIndexOf("a"));  }  }    **public** **class** StringIndexTestExample {  **public** **static** **void** main(String args[]){  *//creating TestString object*  TestString obj = **new** TestString();    *//method call*  obj.indexOfTest();  obj.lastIndexOfTest();  }  } |

## Output:

|  |
| --- |
| 5  12 |

toString() method of Object class is used to provide string representation of an object.

When a object is passed in print() method as an argument then compiler internally call toString() method on the object. It returns object representation as classname@hexadecimal representation of hash code of the object.

## Example:

**ToStringExample1.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of toString method.***  ***\* @author codesjava***  ***\*/***  **class** Student{  String name;  String rollNo;    *//constructor*  Student(String name, String rollNo){  **this**.name = name;  **this**.rollNo = rollNo;  }  }    **public** **class** ToStringExample1 {  **public** **static** **void** main(String args[]){  *//creating Student class object*  Student stu1 = **new** Student("Sunil", "MCA/07/15");  Student stu2 = **new** Student("Sandy", "MCA/07/19");  Student stu3 = **new** Student("Roxy", "MCA/07/32");    *//println internally call toString method*  System.out.println(stu1);  System.out.println(stu2);  System.out.println(stu3);  }  } |

## Output:

|  |
| --- |
| com.javawithease.business.Student@1888759  com.javawithease.business.Student@6e1408  com.javawithease.business.Student@e53108 |

# [How To Write Immutable Class In Java](https://codesjava.com/how-to-write-immutable-class-in-java)

All wrapper classes and String class are immutable classes.

You can write your own immutable class, when creating immutable class just keep in mind that after creating an object of this class object can’t be modified. Any change in existing object result into new object.

1. Make final class so that it cannot inherit.
2. Object state is made up of its properties, declare all properties final. So that its properties value will remain constant.
3. Object properties value can be set using setter methods, so only define getter methods for all properties.

## Example:

**ImmutableClassExample.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to create a immutable class.***  ***\* @author codesjava***  ***\*/***  **final** **class** Student{  *//declare all properties final.*  **final** String rollNo;    **public** Student(String rollNo){  **this**.rollNo = rollNo;  }    *//only create getter method.*  **public** String getRollNo() {  **return** rollNo;  }  }    **public** **class** ImmutableClassExample {  **public** **static** **void** main(String args[]){  *//creating Student object.*  Student obj = **new** Student("MCA/07/06");    System.out.println(obj.getRollNo());  }  } |

## Output:

|  |
| --- |
| MCA/07/06 |

[***Download this example.***](https://codesjava.com/wp-content/uploads/2014/08/StringExample9.rar)

## Advantages/Benefits of immutable class.

1. Objects are thread safe by default.
2. No need to synchronize immutable objects explicitly.

## Disadvantages of immutable classes.

As discussed any change in immutable object result into a new object, hence result in unnecessary garbage.

Substring is a string that is part of a longer string. String class provides the following methods to get a substring from a string.

## *****1. public String substring(int startIndex):*****

Returns a new string which start from a specified string and extends to the end of this string.  It will throw IndexOutOfBoundsException – if startIndex is negative or larger than the length of this String object.

## Example:

**SubStringExample1.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of substring(int startIndex)***  ***\* method to get substrings of a given string.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = "Hello codesjava";    ***/\*\****  ***\* This method is used to get substrings of a given string.***  ***\* @author codesjava***  ***\*/***  **public** **void** showSubString(){  System.out.println(str.substring(6));  }  }    **public** **class** SubStringExample1 {  **public** **static** **void** main(String args[]){  *//creating TestString object.*  TestString obj = **new** TestString();  *//method call*  obj.showSubString();  }  } |

## Output:

|  |
| --- |
| codesjava |

[***Download this example.***](https://codesjava.com/wp-content/uploads/2014/08/StringExample7.rar)

## *****2. public String substring(int startIndex, int endIndex):*****

Returns a new string which start from a specified string and extends to the endIndex – 1 of this string.  It will throw IndexOutOfBoundsException if the startIndex is negative, or endIndex is larger than the length of this string object, or startIndex is larger than endIndex.

## Example:

**SubStringExample2.java**

|  |
| --- |
| ***/\*\****  ***\* This program is used to show the use of***  ***\* substring(int startIndex, int endIndex)***  ***\* method to get substrings of a given string.***  ***\* @author codesjava***  ***\*/***  **class** TestString{  String str = "www.codesjava.com";    ***/\*\****  ***\* This method is used to get substrings of a given string.***  ***\* @author javawithease***  ***\*/***  **public** **void** showSubString(){  System.out.println(str.substring(4,13));  }  }    **public** **class** SubStringExample2 {  **public** **static** **void** main(String args[]){  *//creating TestString object.*  TestString obj = **new** TestString();  *//method call*  obj.showSubString();  }  } |

## Output:

|  |
| --- |
| codesjava |